AUDIT REPORT

DragonX Hybrid

February 2024

Introduction

A time-boxed security review of the **DragonX Hybrid** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **DragonX Hybrid**

DragonX Hybrid is an `ERC721` contract that allows `DragonX` token holders to lock specific amounts and mint NFTs. There are 5 different classes of Dragon NFTs and users can mint any of them if they lock the respective amount of `DragonX` tokens.

Upon minting, users should also provide an amount of `TitanX` tokens and pay a minting fee while during a `burn` call, the burn fee is paid in `DragonX` tokens. The `burnFee` tokens are burned through a separate `DragonBurnProxy` which only functionality is to burn tokens and no native tokens can be sent to it.

NFTs can be minted and burned at any time and can be traded on NFT marketplaces.

Threat Model

Privileged Roles & Actors

- Users - holders of `DragonX` and `TitanX` tokens that can `mint`, `burn`, and trade the NFTs.
- Owner - the only privileged function that can be called by the owner is `setBaseURI()`.

Security Interview

Q: What in the protocol has value in the market?

A: The `DragonX` tokens that are locked in the vault and the NFTs that can be traded and used to unlock the `DragonX` tokens.

Q: In what case can the protocol/users lose money?

A: If an attacker is able to drain the vault with the locked `DragonX` tokens.

Q: What are some ways that an attacker achieves his goals?

A: By obtaining NFT for free (e.g. phishing) and steal the tokens locked in the vault

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 6cf372e20ebd1170a86783f565d5e3aaa0a0062d**

Scope

The following smart contracts were in scope of the audit:

- `DragonBurnProxy.sol`
- `DragonHybrid.sol`

The following number of issues were found, categorized by their severity:

- Critical & High: 0 issues
- Medium: 0 issues
- Low: 0 issues
- Informational: 2 issues

Findings Summary

ID	Title	Severity
[I-01]	Missing event emissions in a state-changing method	Informational
[I-02]	Prefer Solidity Custom Errors over `require` statements with strings	Informational

Detailed Findings

[I-01] Missing event emissions in a state-changing method

It's a best practice to emit events on every state-changing method for off-chain monitoring. Consider emitting one in `setBaseURI()`.

[I-02] Prefer Solidity Custom Errors over `require` statements with strings

Using Solidity Custom Errors has the benefits of less gas spent in reverted transactions, better interoperability of the protocol as clients of it can catch the errors easily on-chain, as well as you can give descriptive names of the errors without having a bigger bytecode or transaction gas spending, which will result in a better UX as well. Consider replacing the `require` statements with custom errors.